中华人民共和国国家知识产权局
STATE INTELLECTUAL PROPERTY OFFICE
OF THE PEOPLE'S REPUBLIC OF CHINA

证 明

CERTIFICATE

本证明之附件是向中国专利局作为受理局提交的下列国际申请副本

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY OF THE BELOW IDENTIFIED INTERNATIONAL APPLICATION THAT WAS FILED WITH THE CHINESE PATENT OFFICE AS RECEIVING OFFICE

国际申请号: INTERNATIONAL APPLICATION NUMBER	PCT/CN2005/002403
国际申请日: INTERNATILNAL FILING DATE	30.12月 2005(30.12.2005)
发明名称: TITLE OF INVENTION	TYPE CHECKING FOR OBJECT-ORIENTED PROGRAMMING LANGUAGES

中华人民共和国国家知识产权局局长
COMMISSIONER OF THE STATE INTELLECTUAL PROPERTY
OFFECE OF THE PEOPLE'S REPUBLIC OF CHINA

田力普

二零零六年五月十七日
MAY 17, 2006

HOME COPY

PCT

REQUEST

The undersigned requests that the present international application be processed according to the Patent Cooperation Treaty.

For receiving Office use only

PCT/CN 2005 / 0 0 2 4 0 3
International Application No.

3 0 · 12月 2005 (3 0 · 1 2 · 2 0 0 5)
International Filing Date

RO/CN 中华人民共和国国家知识产权局 PCT International Application
Name of receiving Office and "PCT International Application"

Applicant's or agent's file reference
(if desired) (12 characters maximum) FPEL05150074

Box No. I TITLE OF INVENTION

TYPE CHECKING FOR OBJECT-ORIENTED PROGRAMMING LANGUAGES

Box No. II APPLICANT □ This person is also inventor

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

INTEL CORPORATION
2200 Mission College Blvd.
Santa Clara, California 95052
United States of America

Telephone No.

Facsimile No.

Teleprinter No.

Applicant's registration No. with the Office

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: □ all designated States ☒ all designated States except the United States of America □ the United States of America only □ the States indicated in the Supplemental Box

Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

GUO, Peng
Room 501 North Building #2
Guanghua Road
Chaoyang District, Beijing 100027
P. R. of China

This person is:

□ applicant only

☒ applicant and inventor

□ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality: CN

State *(that is, country)* of residence: CN

This person is applicant for the purposes of: □ all designated States □ all designated States except the United States of America ☒ the United States of America only □ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on a continuation sheet.

Box No. IV AGENT OR COMMON REPRESENTATIVE; OR ADDRESS FOR CORRESPONDENCE

The person identified below is hereby/has been appointed to act on behalf of the applicant(s) before the competent International Authorities as: ☒ agent □ common representative

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country.)*

China Patent Agent (H.K.) Ltd.
22/F, Great Eagle Centre
23 Harbour Road, Wanchai
Hong Kong Special Administrative Region
The People's Republic of China

Telephone No.
(852)28284688

Facsimile No.
(852)28271018

Teleprinter No.

Agent's registration No. with the Office

□ **Address for correspondence:** Mark this check-box where no agent or common representative is/has been appointed and the space above is used instead to indicate a special address to which correspondence should be sent.

See Notes to the request form

Sheet No. ...2...

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

If none of the following sub-boxes is used, this sheet should not be included in the request.

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

SHI, Xiaohua
1-1004# Taiyangyuan
Haidian District
Beijing 100086
P. R. of China

This person is:
□ applicant only
☒ applicant and inventor
□ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality: CN

State *(that is, country)* of residence: CN

This person is applicant for the purposes of: □ all designated States □ all designated States except the United States of America ☒ the United States of America only □ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
□ applicant only
□ applicant and inventor
□ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality:

State *(that is, country)* of residence:

This person is applicant for the purposes of: □ all designated States □ all designated States except the United States of America □ the United States of America only □ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
□ applicant only
□ applicant and inventor
□ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality:

State *(that is, country)* of residence:

This person is applicant for the purposes of: □ all designated States □ all designated States except the United States of America □ the United States of America only □ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
□ applicant only
□ applicant and inventor
□ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality:

State *(that is, country)* of residence:

This person is applicant for the purposes of: □ all designated States □ all designated States except the United States of America □ the United States of America only □ the States indicated in the Supplemental Box

□ Further applicants and/or (further) inventors are indicated on another continuation sheet.

See Notes to the request form

Box No. V DESIGNATIONS

The filing of this request constitutes under Rule 4.9(a), the designation of all Contracting States bound by the PCT on the international filing date, for the grant of every kind of protection available and, where applicable, for the grant of both regional and national patents.

However,

□ DE Germany is not designated for any kind of national protection

□ KR Republic of Korea is not designated for any kind of national protection

□ RU Russian Federation is not designated for any kind of national protection

(The check-boxes above may be used to exclude (irrevocably) the designations concerned in order to avoid the ceasing of the effect, under the national law, of an earlier national application from which priority is claimed. See the Notes to Box No. V as to the consequences of such national law provisions in these and certain other States.)

Box No. VI PRIORITY CLAIM

The priority of the following earlier application(s) is hereby claimed:

Filing date of earlier application *(day/month/year)*	Number of earlier application	Where earlier application is:		
		national application: country or Member of WTO	regional application:* regional Office	international application: receiving Office
item (1)				
item (2)				
item (3)				

□ Further priority claims are indicated in the Supplemental Box.

The receiving Office is requested to prepare and transmit to the International Bureau a certified copy of the earlier application(s) *(only if the earlier application was filed with the Office which for the purposes of this international application is the receiving Office)* identified above as:

□ all items □ item (1) □ item (2) □ item (3) □ other, see Supplemental Box

* *Where the earlier application is an ARIPO application, indicate at least one country party to the Paris Convention for the Protection of Industrial Property or one Member of the World Trade Organization for which that earlier application was filed (Rule 4.10(b)(ii)):* ....

Box No. VII INTERNATIONAL SEARCHING AUTHORITY

Choice of International Searching Authority (ISA) *(if two or more International Searching Authorities are competent to carry out the international search, indicate the Authority chosen; the two-letter code may be used)*:

ISA / CN...

Request to use results of earlier search; reference to that search *(if an earlier search has been carried out by or requested from the International Searching Authority)*:

Date *(day/month/year)* Number Country *(or regional Office)*

Box No. VIII DECLARATIONS

The following declarations are contained in Boxes Nos. VIII (i) to (v) *(mark the applicable check-boxes below and indicate in the right column the number of each type of declaration)*:

			Number of declarations
□	Box No. VIII (i)	Declaration as to the identity of the inventor	:
□	Box No. VIII (ii)	Declaration as to the applicant's entitlement, as at the international filing date, to apply for and be granted a patent	:
□	Box No. VIII (iii)	Declaration as to the applicant's entitlement, as at the international filing date, to claim the priority of the earlier application	:
□	Box No. VIII (iv)	Declaration of inventorship (only for the purposes of the designation of the United States of America)	:
□	Box No. VIII (v)	Declaration as to non-prejudicial disclosures or exceptions to lack of novelty	:

Sheet No. ....4....

Box No. IX CHECK LIST; LANGUAGE OF FILING

This international application contains:

(a) **in paper form, the following number of sheets:**

request (including declaration sheets)	:	4
description (excluding sequence listing and/or tables related thereto)	:	15
claims	:	8
abstract	:	1
drawings	:	4
Sub-total number of sheets	:	32
sequence listing	:	
tables related thereto	:	
(for both, actual number of sheets if filed in paper form, whether or not also filed in computer readable form; see (c) below)		
Total number of sheets	:	32

(b) □ **only in computer readable form** (Section 801(a)(i))

(i) □ sequence listing

(ii) □ tables related thereto

(c) □ **also in computer readable form** (Section 801(a)(ii))

(i) □ sequence listing

(ii) □ tables related thereto

Type and number of carriers (diskette, CD-ROM, CD-R or other) on which are contained the

□ sequence listing: ....

□ tables related thereto: ....

(additional copies to be indicated under items 9(ii) and/or 10(ii), in right column)

This international application is **accompanied by** the following item(s) *(mark the applicable check-boxes below and indicate in right column the number of each item)*:

		Number of items
1. ☒	fee calculation sheet	: 1
2. ☒	original separate power of attorney	: 1
3. □	original general power of attorney	:
4. □	copy of general power of attorney; reference number, if any:	:
5. □	statement explaining lack of signature	:
6. □	priority document(s) identified in Box No. VI as item(s):	:
7. □	translation of international application into *(language)*:	:
8. □	separate indications concerning deposited microorganism or other biological material	:
9. □	sequence listing in computer readable form *(indicate type and number of carriers)*	
(i) □	copy submitted for the purposes of international search under Rule 13*ter* only (and not as part of the international application)	:
(ii) □	*(only where check-box (b)(i) or (c)(i) is marked in left column)* additional copies including, where applicable, the copy for the purposes of international search under Rule 13*ter*	:
(iii) □	together with relevant statement as to the identity of the copy or copies with the sequence listing mentioned in left column	:
10. □	tables in computer readable form related to sequence listing *(indicate type and number of carriers)*	
(i) □	copy submitted for the purposes of international search under Section 802(b-*quater*) only (and not as part of the international application)	:
(ii) □	*(only where check-box (b)(ii) or (c)(ii) is marked in left column)* additional copies including, where applicable, the copy for the purposes of international search under Section 802(b-*quater*)	:
(iii) □	together with relevant statement as to the identity of the copy or copies with the tables mentioned in left column	:
11. □	other *(specify)*:	:

Figure of the drawings which should accompany the abstract:

Language of filing of the international application: EN

Box No. X SIGNATURE OF APPLICANT, AGENT OR COMMON REPRESENTATIVE

Next to each signature, indicate the name of the person signing and the capacity in which the person signs (if such capacity is not obvious from reading the request).

CHINA PATENT AGENT (HONG KONG) LIMITED
中国专利代理(香港)有限公司
For Patent Affairs Only
专利申请专用章

For receiving Office use only

1. Date of actual receipt of the purported international application: 30.-12月 2005 (3 0 · 1 2 · 2 0 0 5)

2. Drawings:

□ received:

□ not received:

3. Corrected date of actual receipt due to later but timely received papers or drawings completing the purported international application:

4. Date of timely receipt of the required corrections under PCT Article 11(2):

5. International Searching Authority (if two or more are competent): ISA /

6. □ Transmittal of search copy delayed until search fee is paid

For International Bureau use only

Date of receipt of the record copy by the International Bureau:

This sheet is not part of and does not count as a sheet of the international application.

PCT

FEE CALCULATION SHEET
Annex to the Request

For receiving Office use only

PCT/CN 2005 / 002403
International Application No.

30·12月2005 (30·12·2005)
Date stamp of the receiving Office

Applicant's or agent's file reference: FPEL05150074

Applicant
INTEL CORPORATION etc.

CALCULATION OF PRESCRIBED FEES

1. TRANSMITTAL FEE	CNY500	T	CNY500
2. SEARCH FEE	CNY1500	S	CNY1500

International search to be carried out by CN

(If two or more International Searching Authorities are competent to carry out the international search, indicate the name of the Authority which is chosen to carry out the international search.)

3. INTERNATIONAL FILING FEE

Where items (b) and/or (c) of Box No. IX apply, enter **Sub-total number of sheets**
Where items (b) and (c) of Box No. IX do not apply, enter **Total number of sheets** } 32

i1	first 30 sheets	CHF1400	i1	CHF1400
i2	2 (number of sheets in excess of 30) x CHF15 (fee per sheet) =	CHF30	i2	CHF30

i3 additional component (only if sequence listing and/or tables related thereto are filed in computer readable form under Section 801(a)(i), or both in that form and on paper, under Section 801(a)(ii)):

400 x ______ (fee per sheet) = ______ i3

Add amounts entered at i1, i2 and i3 and enter total at I: CHF1 430 I — CHF1430

(Applicants from certain States are entitled to a reduction of 75% of the international filing fee. Where the applicant is (or all applicants are) so entitled, the total to be entered at I is 25% of the international filing fee.)

4. FEE FOR PRIORITY DOCUMENT *(if applicable)* ______ P

5. TOTAL FEES PAYABLE: CNY2000CHF1430 TOTAL — CNY2000 CHF1430
Add amounts entered at T, S, I and P, and enter total in the TOTAL box

MODE OF PAYMENT

☒ authorization to charge deposit account (see below) ☐ postal money order ☒ cash ☐ coupons
☐ cheque ☐ bank draft ☐ revenue stamps ☐ other *(specify)*:

AUTHORIZATION TO CHARGE (OR CREDIT) DEPOSIT ACCOUNT
(This mode of payment may not be available at all receiving Offices)

☒ Authorization to charge the total fees indicated above.

☒ *(This check-box may be marked only if the conditions for deposit accounts of the receiving Office so permit)* Authorization to charge any deficiency or credit any overpayment in the total fees indicated above.

☒ Authorization to charge the fee for priority document.

Receiving Office: RO/ CN
Deposit Account No.:
Date: 12/30/2005
Name:
Signature:

CHINA PATENT AGENT (HONG KONG) LIMITED
中国专利代理(香港)有限公司
For Patent Affairs Only
专利申请专用章
02

See Notes to the fee calculation sheet

TYPE CHECKING FOR OBJECT-ORIENTED PROGRAMMING LANGUAGES

BACKGROUND

[0001] Object-oriented programming languages may support inheritance, which may use an existing type to derive a new type. Derived types may inherit data and operations of super-type of the derived types; and they may overwrite existing operations or add new ones. Complex object-oriented programs may contain complex inheriting hierarchies. These hierarchies may often require that the program explicitly convert an object reference from one type to another type. This type of conversion may need run-time type checking that may be used to check whether the object is cast into an invalid target type. For most implementation of object-oriented programming languages, besides a memory to store fields of an object, each object may have an object header to provide basic services of object-oriented programming, such as class hierarchies with virtual methods, and other metadata that may be tapped into for different kinds of uses.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] The invention described herein is illustrated by way of example and not by way of limitation in the accompanying figures. For simplicity and clarity of illustration, elements illustrated in the figures are not necessarily drawn to scale. For example, the dimensions of some elements may be exaggerated relative to other elements for clarity. Further; where considered appropriate, reference labels have been repeated among the figures to indicate corresponding or analogous elements.

[0003] FIG. 1 illustrates an embodiment of a system that may implement type checking.

[0004] FIG. 2 illustrates an embodiment of a data structure that may be used in the present invention.

[0005] FIG. 3 illustrates an embodiment of a *checkcast* process that may be used in type checking for an object-oriented programming language.

[0006] FIG. 4 illustrates an embodiment of an *instanceof* process that may be used in type checking for an object-oriented programming language.

[0007] FIG. 5 is a block diagram illustrating an embodiment of a system that may dynamically generate type checking code in a Java run-time environment.

[0008] FIG. 6 is a schematic diagram of a type checking related process according to an embodiment of the present invention.

DETAILED DESCRIPTION

[0009] The following description describes techniques to accelerate run-time type checking in Java virtual machine. The implementation of the techniques is not restricted in Java virtual machine; it may be used by any execution environments for similar purposes. In the following description, numerous specific details such as logic implementations, opcodes, means to specify operands, resource partitioning/sharing/duplication implementations, types and interrelationships of system components, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the present invention. However, the invention may be practiced without such specific details. In other instances, control structures and full software instruction sequences have not been shown in detail in order not to obscure the invention.

[0010] References in the specification to "one embodiment", "an embodiment", "an example embodiment", etc., indicate that the embodiment described may include a particular feature, structure, or characteristic, but every embodiment may not necessarily include the particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to effect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described.

[0011] Embodiments of the invention may be implemented in hardware, firmware, software, or any combination thereof. Embodiments of the invention may also be implemented as instructions stored on a machine-readable medium, which may be read and executed by one or more processors. A machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computing device). For example, a machine-readable medium may include read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.), and others.

[0012] FIG. 1 illustrates an example system that may implement the methods of the present invention. Referring to FIG. 1, in one embodiment, system 100 may comprise a processor 110. Processor 110 may coupled with a memory 120 via a bus 115. Processor 110 may be any type of processor adapted to execute instructions from memory 120. For example, processor 110 may be a microprocessor, a digital signal processor, a microcontroller, or the like.

[0013] In one embodiment, processor 110 and memory 120 may be included on an integrated circuit board, and bus 115 may be implemented using traces on the circuit board. In another embodiment, processor 110 and memory 120 may be included within the same integrated circuit, and bus 115 may be implemented using interconnect within the integrated circuit.

[0014] As shown in FIG. 1, in one embodiment, processor 110 may execute a first compiler 130. The first complier 130 may comprise any type of compiler adapted to output instructions for type checking. For example, the first compiler 130 may comprise a Java compiler that may output Java byte codes for runtime checking. In another embodiment, processor 110 may execute a Java Virtual Machine (JVM) 140. For example, JVM 140 may be implemented with a Just-In-Time (JIT) compiler. JVM 140 may be coupled with the first compiler 130 to translate the Java byte codes outputted from the first complier 130 to architecture specific native codes. In yet another embodiment, the first compiler 130 and/or JVM 140 may be implemented with software that may be stored in memory 120. In one embodiment, processor 110 may perform operations according to the instructions from JVM 140, for example, a process as shown in FIG. 3 or 4.

[0015] FIG. 3 illustrates a flowchart showing an embodiment of a method that may be used in type checking for an object-oriented programming language. The following description may be focused on Java that is one of the object-oriented programming languages; however, other embodiments may be used for other dynamically compiled object-oriented programming languages such as, for example, C#, Python, LISP, etc. In the example of Java, it may have one or more, e.g., five, conversion contexts in which conversion expression may occur. For example, an example conversion from type S to type T may allow an expression of Type S to be treated at compiling time as if it had type T instead. In some cases, to achieve type safety during conversion, a corresponding action may be made at run time to check the validity of the conversion. For example, for an object, a conversion from type "class Object" to type "class Thread" may need a run-time check to see if the object is an instance of class Thread or one of its subclass. In one embodiment, Java Virtual Machine 140 may use one or more byte codes or instructions, e.g., *checkcast* and *instanceof*, for performing type checking on an object.

[0016] Referring to FIG. 3, the method may be used for a *checkcast* process. In block 302, processor 110 may obtain an object header. In one embodiment, processor 110 may get the object header from an object handle of an object 200 as shown in FIG. 2. Referring to FIG. 2, in one embodiment, object header 205 may comprise one or more bits that may represent the class of the object. Numerical reference 220 may represent a data structure. For example, data structure 220 may be implemented as table, array, list or the like. In one embodiment, data structure 220 may comprise a target class table. In one embodiment, data structure 220 may comprise one or more entries that each may comprise a target class handle associated with a hotspot. In another embodiment, the data structure 220 may be dedicated to one object, or shared by one or more objects inside JVM 140. For example, numerical reference 230 may refer to a second object. The second object 230 may comprise a second object header 235 that may be obtained from a second object handle 2; however, other embodiments may comprise a different number of objects.

[0017] For example, processor 110 may use dynamic profiling to find out one or more hotspots, e.g., during program running. For example, a hotspot may refer to a hot check point that may be a location in a program at which the type checking operation from objects to a given target class occurs one or more times, e.g., a lot of times, during program running. The processor 110 may deduce a target class from a type checking hotspot. In one embodiment, target class handle 222 may contain class identification of a first target class; target class handle 224 may contain class identification of a second target class; and target class handle 226 may contain class identification of the nth target class; however, other embodiments may identify target classes with different information. In one embodiment, the method of FIG. 3 may be used for type checking associated with a hotspot. In another embodiment, one or more type checking hotspots with the same target class may share the same bit indicator of an object header and thus the same target class entry.

[0018] In one embodiment, the first object header 205 may allocate one or more bits 210 that each may correspond to an entry in data structure 220. Similarly, the second object header 235 may allocate each of one or more bits 240 to a corresponding entry in data structure 220. For example, bit 212 may be allocated to the first entry 222; bit 214 may be assigned to the second entry 224; and bit 242 may be assigned to the nth entry 226; however, other embodiments may allocate a bit for an entry in a different order. In one embodiment, JVM 140 may assert the one or more bits 210 in the first object header 205 and/or one or more bits 240 in the second object header 235 as indicators that may each indicate whether a type checking associated with a corresponding entry is successful. For example, the one or more bit indicators 210 and/or 240 may have a first logic value (for example, "0") in their initial states. Processor 110 may assert a bit indicator 210 and/or 240 corresponding to an entry to a second logic value (for example, "1"), in response to determining that a type checking associated with the entry is successful.

[0019] In block 304, during performing a *checkcast* type checking for an object at a hotspot, processor 110 may check whether a bit indicator 210 in object header 205 of the object is asserted. For example, processor 110 may determine whether the bit indicator 210 that is allocated to a target class of the hotspot is asserted to a second logic value (for example, "1"). In response to the bit indicator 210 being asserted, processor 110 may determine that the type checking between the object class and the target class associated with the hotspot is successful and may terminate the type checking at the hotspot.

[0020] On the contrary, in block 306, JVM 140 may call a function that may be stored in memory 120 to perform a type checking for the object class, in response to determining that the bit indicator 210 is deasserted. For example, processor 110 may call the function, in response to determining that the bit indicator 210 has a first logic value or the bit indicator 210 is in its initial state. In one embodiment, the function may comprise a special checkcast helper, for example, with reference to blocks 308, 310, 312, 314, 316 and 318. In one embodiment, processor 110 may determine whether the object class of the current object and the target class match a predetermined criterion/condition.

[0021] In one embodiment, processor 110 may traverse the super classes of the class of the current object to see whether one of the super classes could be the same as the target class. For example, the type checking for the class of the current object is successful, in response to the processor 110 determining that one of the super classes is the same as the target class. In another embodiment, processor 110 may traverse a class hierarchy associated with the class of the current object to determine whether the target class is represented in the class hierarchy based on one or more predetermined criteria or conditions. For example, the processor 110 may determine that the type checking between the class of the current object and the target class is successful, in response to determining that the target class is represented in the class hierarchy.

[0022] In one embodiment, the class hierarchy may be implemented as an array of class references or the like that may be used to perform type checking in Java computing environments. In one embodiment, the class hierarchy may represent all the parent classes of Java classes in a hierarchical relationship. In another embodiment, an example of a predetermined criterion/condition is described as follows, wherein S=class of an object; T=target class:

I) If S is a nonarray class, then:

i) If T is a class type, then S must be the same class as T, or a subclass of T.

ii) If T is an interface type, then S must implement interface T.

II) If S is an interface type, then:

i) If T is a class type, then T must be Object.

ii) If T is an interface type, then T must be the same interface as S or a super interface of S.

III) If S is a class representing the array type SC[], then:

i) If T is a class type, then T must be Object.

ii) If T is an array type TC[], then one of the following must be true:

a) TC and SC are the same primitive type.

b) TC and SC are reference types, and type SC can be cast to TC by recursive application of these criteria.

iii) If T is an interface type, T must be one of the interfaces implemented by arrays.

[0023] In block 310, processor 110 may assert a bit indicator 210 in the object header of the current object to indicate that the type checking between the object class and the target class is successful. For example, processor 110 may assert the bit indicator 210 to a second logic value. In one embodiment, the bit indicator 210 may correspond to an entry in data structure 220 that represents a target class, for example, class handle, associated with a hotspot. Further, in block 312, processor 110 may return the result indicating that the type checking between the object class and the target class is successful or the object class and the target class matches a predetermined criterion/condition. For example, processor 110 may return "True". In another embodiment, processor 110 may return a "True" result before asserting the bit indicator 210. Conversely, in response to determining that the class of the current object and the target class do not match the criterion/condition or the type checking fails, processor 110 may return a corresponding result, for example, "False" (block 314).

[0024] In block 316, processor 110 may determine whether the object has passed the type checking based upon the result returned in blocks 312 or 314. For example, in response to determining that the object has not passed the type checking based on a result "False" or a failure result, processor 110 may throw an exception (block 318). Conversely, in response to a pass result, e.g., "True", the type checking flow of FIG.3 is ended. In one embodiment, processor 110 may continue other operations of the system 100.

[0025] FIG. 4 is a flowchart illustrating a method according to another embodiment of the present invention. Referring to FIG. 4, the process of FIG. 4 is similar to that of FIG. 3 except blocks 406, 418 and 420. In particular, in block 406, processor 110 may call a function that may be stored in memory 120, in response to determining that a bit indicator 210 associated with a target class of a type checking hotspot has not been asserted. For example, processor 110 may call a special *instanceof* helper. In another embodiment, in block 418, processor 110 may push a first code, e.g., a

True code, on an operand stack of memory 120 in response to determining that the object has passed the type checking. Conversely, in block 420, processor 110 may push a second code, e.g., a False code, on the operand stack in response to determining that the object has not passed the type checking.

[0026] Similar with 80-20 rule, the hot or hottest check points (using, e.g., either *checkcast* or *instanceof* bytecode), that consume most runtime on type checking may occur at several points (thus for several target classes). And, type checkings at one or more or most of the hot or hottest check points may be successful, e.g., in most cases. In one embodiment, the method as shown in FIG. 3 or 4 may be used for every such check point. For example, the JVM 140 may use dynamic profiling to find out hot check points that are either *checkcast* or *instanceof* bytecode, during the processor executing the Java program, in which the type checking are made using conventional methods.

[0027] While the methods of FIGs. 3 and 4 are illustrated as a sequence of operations, the illustrated operations may be performed in a different order in other embodiments.

[0028] FIG. 5 is a block diagram illustrating an embodiment of a system 500 that may execute Java program in, e.g., system 100. In one embodiment, the system 500 may be a Java runtime system. The system 500 may contain a dynamic compiler 534 that may translate the Java program's bytecode into instructions of the processor 110 that may be executed directly on processor 110. In one embodiment, the dynamic compiler 534 may dynamically generate type checking code.

[0029] As shown in FIG. 5, system 500 may comprise a first compiler 510 that may compile Java source code 502 to Java byte code 504. In one embodiment, the first compiler 510 may be itself a computer program. Referring to FIG. 5, a loader 532 may load the byte codes 504 into JVM 530. A dynamic compiler 534 in JVM 530 may turn the byte code 504 loaded via loader 532 into native code 508 associated with the program. In one embodiment, the dynamic compiler 534 may comprise a "just-in-time" (JIT) compiler or the like. In another embodiment, JVM 530 may comprise an interpreter to interpret the byte codes 504. In another embodiment, JVM 530 may comprise a profiler 536 that may perform dynamic profiling during the program running to detect one or more hot check points, i.e., hotspots, that may occur at several points (thus for several type classes) in a program. Profiler 536 may return the detected hotspots 506 to dynamic compiler 534.

[0030] FIG. 6 is a schematic diagram of a type checking related process 600 that may be generated by JVM 530. In one embodiment, block 610 may refer to the native instructions of processor 110 translated from Java bytecodes 504. In one embodiment, dynamic compiler 534 may use normal methods to generate type checking code at the first time compilation. For example, dynamic compiler 534 may generate type checking code to call normal *checkcast* helpers 612 and 616 and a normal *instanceof* helper 614 for the type checking; however, other embodiments may use different functions for type checking in a different order. The normal type checking helper may get a class of an object and traverse a class hierarchy to check whether the object class and target class meets a criteria, which may be time consuming. During program running based on type checking code generated during the first time compilation, profiler 536 may perform dynamic profiling to detect type checking hotspots, which may trigger a second time compilation for these hotspots in dynamic compiler 534. Referring to FIG. 6, profiler 536 may determine a hotspot 622 of the *checkcast* helper 612 and a hotspot 624 of the *instanceof* helper 614.

[0031] In one embodiment, during the second time compilation, dynamic compiler 534 may deduce a target class for a hotspot of *checkcast* or *instanceof* helper. For example, dynamic compiler 534 may deduce a target class for hotspots 622 and 624 that relates to a normal *checkcast* helper and a normal *instanceof* helper, respectively. Dynamic compiler 534 may allocate an entry in data structure 220 to save the target class handle of the deduced target class and a corresponding bit indicator in an object for the entry. Referring to FIG. 2, in one embodiment, one bit indicator 210 may comprise a bit in object header 205 of an object 200 (the bit may be cleared by default). Dynamic compiler 534 may regenerate type checking code (for example, block 630) to check the bit indicator 210 and call a special helper for the hotspot. In one embodiment, for hotspot 622, dynamic compiler 534 may regenerate type checking code to check whether a bit indicator associated with hotspot 622 is asserted and call a special *checkcast* helper 632. For hotspot 624, dynamic compiler 534 may regenerate type checking code to check whether a bit indicator associated with hotspot 624 is asserted and call a special *instanceof* helper 632. In one embodiment, a special helper may be different from a normal helper in that the special helper may assert a bit indicator in an object header of an object in case of type checking success, for example, as shown in FIGS. 3 and 4. Comparing with traversing the class hierarchy, checking a bit indicator in an object header may reduce the runtime type checking overhead. In another embodiment, one or more type checking hotspots with the same target class may share the same bit indicator and the same target class entry.

[0032] In the following, an embodiment of a program that may need runtime type checking is described. For example, it may be assumed that there are two classes, Parent and Child. The program, e.g., Java program, may use the following statements to define the two classes.

```
class Parent {
   void foo() { }
}
class Child extends Parent {
   void foo() { }
   void bar() { }
}
```

[0033] In one embodiment, the program may use the following statement to create an instance of Child and assign to an reference (similar to a pointer) which has type Parent:

```
Parent ref = new Child();
```

[0034] In another embodiment, the program may do type casting and call Child's bar as follows:

```
Child ref2 = (Child) ref;
      ref2.bar();
```

[0035] The first compiler 130 may output the bytecode *checkcast*, for example, before assigning the *ref* to *ref2* to do runtime checking so as to make sure the casting is safe (e.g., the object is an instance of the target class or the target class's sub class).

[0036] In another embodiment, in the implementation 500, JVM 530 may be implemented with dynamic compiler 534, such as a Just-In-Time (JIT) compiler. For example, the dynamic compiler 534 may translate the byte codes from the first compiler 510 to architecture native codes that may be recognized by processor 110. For *checkcast & instanceof*, dynamic compiler 534 may emit a call to a JVM 530's helper function, e.g., *checkcast* helper or *instanceof* helper. Processor 110 may use the function to do type checking and decide whether to throw an exception (for *checkcast*) or return the check status (for *instanceof*). An example of bytecodes and translated native instructions may be as follows:

```
checkcast Child
    ldr r0, [sp]          // get the object reference
    mov r1, #0x5a3200     // the target class
    bl 0x69d268           // call checkcast helper with object reference and target class
```

[0037] In the example of a Java program, sometimes only several type checking may be hot. After dynamic compiler 534 completes the first time compilation, JVM 530 may use dynamic profiling to find hot places or hotspots, for example, in profiler 536. Another optimized procedure may be used for these hot places by saving the type checking results of the hot places into an object header 200. In one embodiment, dynamic compiler 534 may regenerate check codes for the hotspots. For example, if bit 0 in the object header is assigned for a hotspot, the following codes may be generated:

```
checkcast Child
    ldr r0, [sp]          // get object reference
    ldr r2, [r0]          // get object header (the header has the information on object class, etc.)
    tst r2, #0x1          // check if bit 0 is asserted
```

```
bne check_done      // if asserted, then this object has been checked in this site, and the previous check is successful,
                    // skip the helper call
mov r1, #0x5a3200   // the target class
bl 0x69d268         // call check helper with object reference and target class
check_done:
...
```

[0038] While certain features of the invention have been described with reference to embodiments, the description is not intended to be construed in a limiting sense. Various modifications of the embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.

What is claimed is:

1. A method comprising

getting an object header from an object, and

checking from the object header a result of a first time type checking between a class of the object and a target class specified by a hotspot in the first time type checking.

2. The method of claim 1 further comprising

determining whether the object header comprises an indicator that is asserted to indicate a first time type checking success between the object class and the target class associated with the indicator.

3. The method of claim 1 further comprising

determining whether the object header comprises an indicator that is deasserted to indicate a first time type checking failure between the object class and the target class associated with the indicator.

4. The method of claim 1 further comprising

skipping a second time type checking between the object class and the target class, in response to determining that the object header comprises an indicator that is asserted to indicate a first time type checking success.

5. The method of claim 1 further comprising

performing a second time type checking between the object class and the target class, in response to determining that the object header comprises an indicator that is deasserted to indicate a first time type checking failure.

6. The method of claim 1 further comprising

detecting the hotspot in the first time type checking by dynamic profiling.

7. A system, comprising

a processor to get an object header from an object, and obtain from the object header a result of a first time type checking at a hotspot between a class of the object and a target class specified by the hotspot; and

a memory to save the target class.

8. The system of claim 7, wherein the processor further to

determine that the first time type checking at the hotspot is successful, in response to detecting that the object header comprises an indicator associated with the target class that has a first logic value.

9. The system of claim 7, wherein the processor further to

perform a second time type checking between the object class and the target class, in response to detecting that an indicator associated with the target class in the object header has a second logic value.

10. The system of claim 7, wherein the processor further to

traverse a class hierarchy associated with the class of the object, in response to determining that the first time type checking at the hotspot is failed.

11. The system of claim 7, wherein the processor further to

assert an indicator associated with the target class in the object header, in response to determining in a second time type checking at the hotspot that the class of the object and the target class match a type checking condition.

12. The system of claim 7, wherein the processor further to

return a signal indicating that the type checking is successful, in response to determining that the class of the object and the target class match a predetermined criterion.

13. The system of claim 7, wherein the memory further to save a beginning address of a handle of the target class, and wherein the processor further to detecting the hotspot by dynamic profiling.

14. A machine readable medium comprising a plurality of instructions that in response to being executed result in a computing device

obtaining an object header from an object, and

checking a bit indicator in the object header to indicate a result of a first time type checking at a hotspot between a class of the object and a target class specified by the hotspot.

15. The machine readable medium of claim 14, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

skipping a second time type checking at the hotspot between the object class and the target class, in response to determining that the bit indicator is asserted to indicate a successful result.

16. The machine readable medium of claim 14, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

performing a second time type checking at the hotspot between the object class and the target class, in response to determining that the bit indicator is deasserted to indicate a failure result.

17. The machine readable medium of claim 14, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

detecting at the hotspot whether the object class and the target class match a type checking criterion, in response to determining that the bit indicator indicates a failure result.

18. The machine readable medium of claim 16, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

asserting the bit indicator, in response to determining in the second time type checking that the object class and the target class match a type checking criterion.

19. The machine readable medium of claim 16, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

returning a signal indicating a result of the second time type checking.

20. The machine readable medium of claim 16, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

throwing an exception, in response to determining that the second time type checking is successful.

21. The machine readable medium of claim 16, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

pushing a result code on a stack to indicate whether the type checking is successful.

22. A system comprising,

a compiler to convert source code associated with a first time type checking for an object into byte code;

a loader coupled with the compiler to load the byte code;

a dynamic compiler coupled with the loader to receive the byte code from the loader, and to generate first native code associated with the first time type checking based on the byte code; and

a profiler coupled with the dynamic compiler to detect a hotspot in the first time type checking based on the first native code, and to return the hotspot to the dynamic compiler.

23. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to determine a type checking result between a class of the object and a target class specified by the hotspot from an object header of the object.

24. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to detect a type checking success between a class of the object and a target class specified by the hotspot, in response to determining that an indicator associated with the target class in an object header of the object has a first logic value.

25. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to detect a type checking failure between a class of the object and a target class specified by the hotspot, in response to determining that an indicator associated with the target class in an object header of the object has a second logic value.

26. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to skip type checking between a class of the object and a target class specified by the hotspot, in response to determining that an object header of the object indicates a type check success.

27. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to perform type checking between a class of the object and a target class specified by the hotspot, in response to determining that an indicator associated with the target class in an object header of the object is deasserted.

28. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to assert an indicator in an object header of the object, in response to a type checking success between a class of the object and a target class specified by the hotspot.

29. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to traverse super classes of a class of the object to determine whether one of the super classes is the same as a target class as specified by the hotspot, in response to determining a type checking failure between the class of the object and the target class from an object header of the object.

30. The system of claim 22, wherein the dynamic compiler further to regenerate second native code that calls a type checking function for the hotspot to traverse a class hierarchy associated with a class of the object to determine whether a target class as specified by the hotspot is represented in the class hierarchy, in response to determining a type checking failure between the class of the object and the target class from an object header of the object.

ABSTRACT

Type checking between an object class and a target class may comprise getting an object header from an object, and checking from the object header a result of a first time type checking between a class of the object and a target class specified by a hotspot in the first time type checking.

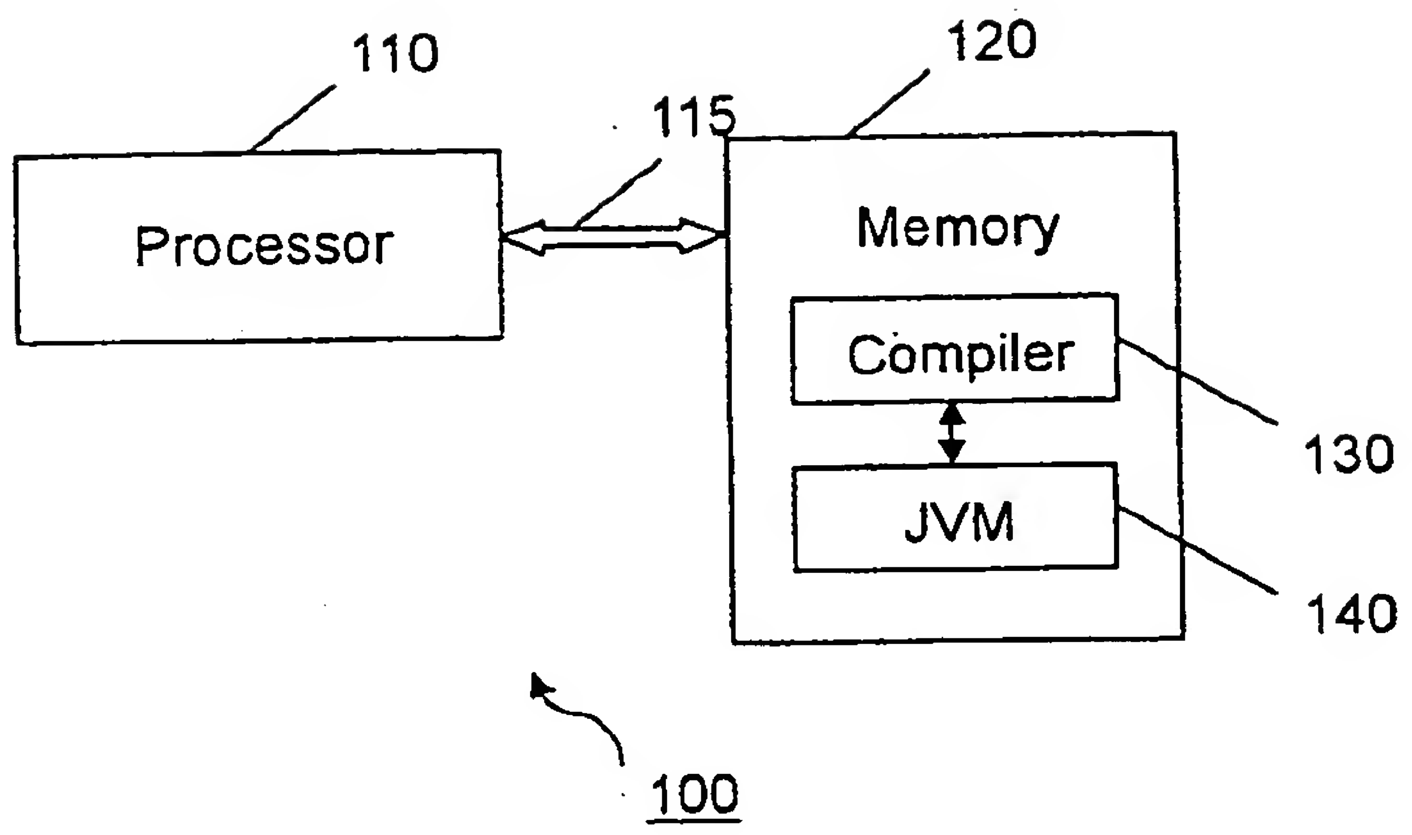
110
115
120
Processor
Memory
Compiler
JVM
130
140
100

FIG. 1

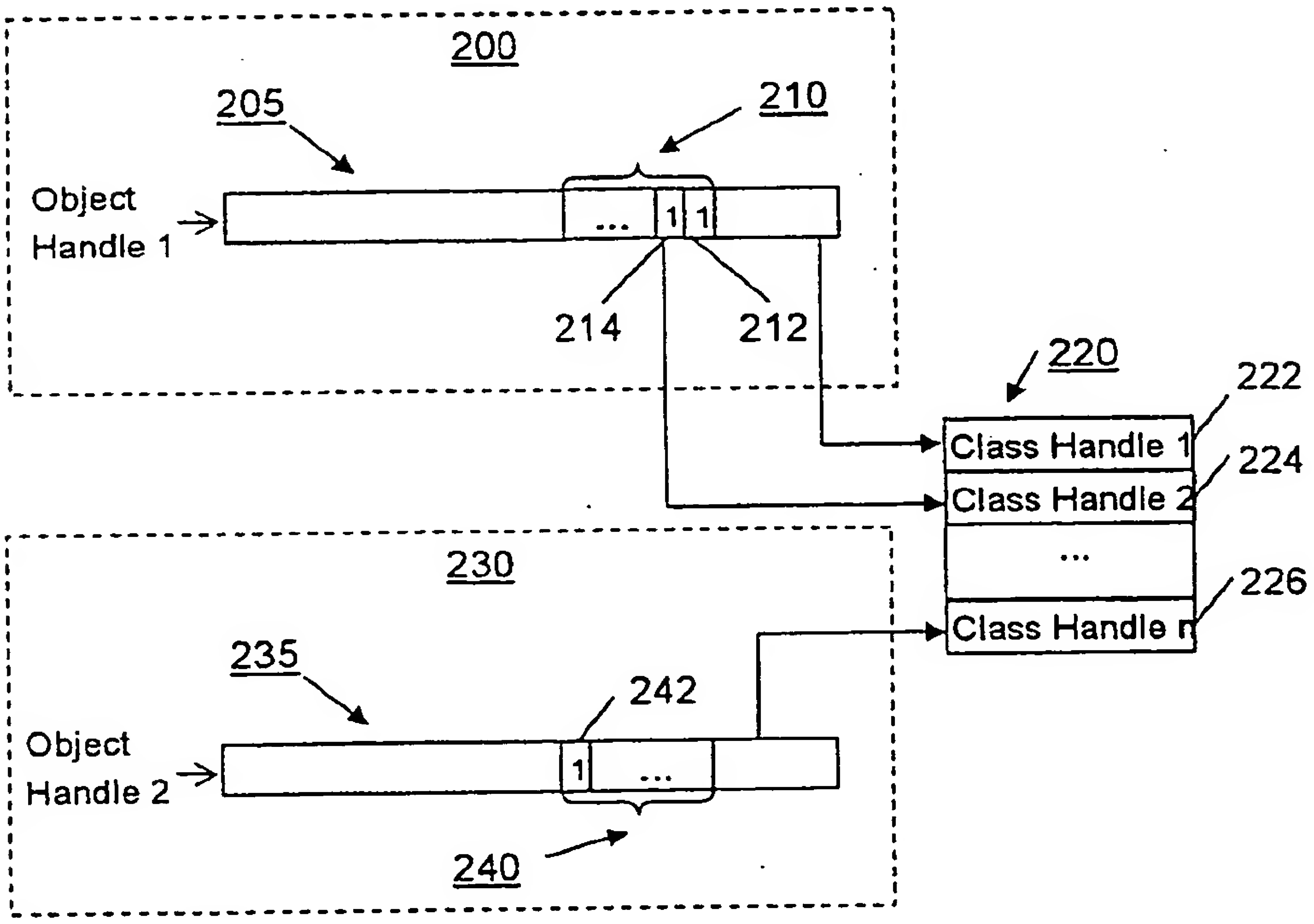
200
205
210
Object Handle 1
...
1
1
214
212
220
222
Class Handle 1
224
Class Handle 2
...
226
Class Handle n
230
235
242
Object Handle 2
1
...
240

FIG. 2

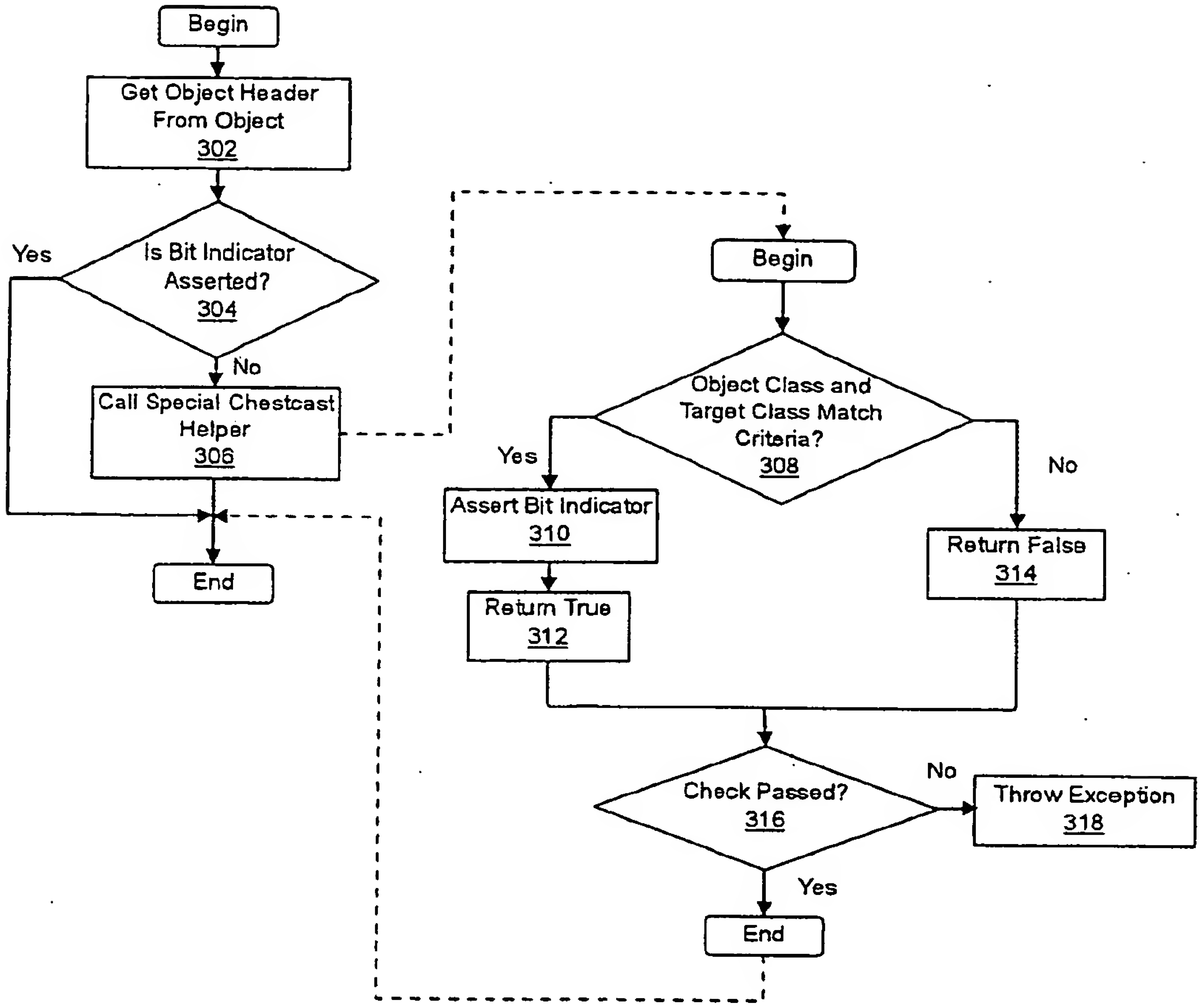
Begin
Get Object Header From Object 302
Is Bit Indicator Asserted? 304
Yes
No
Call Special Chestcast Helper 306
End
Begin
Object Class and Target Class Match Criteria? 308
Yes
No
Assert Bit Indicator 310
Return True 312
Return False 314
Check Passed? 316
No
Throw Exception 318
Yes
End

FIG. 3

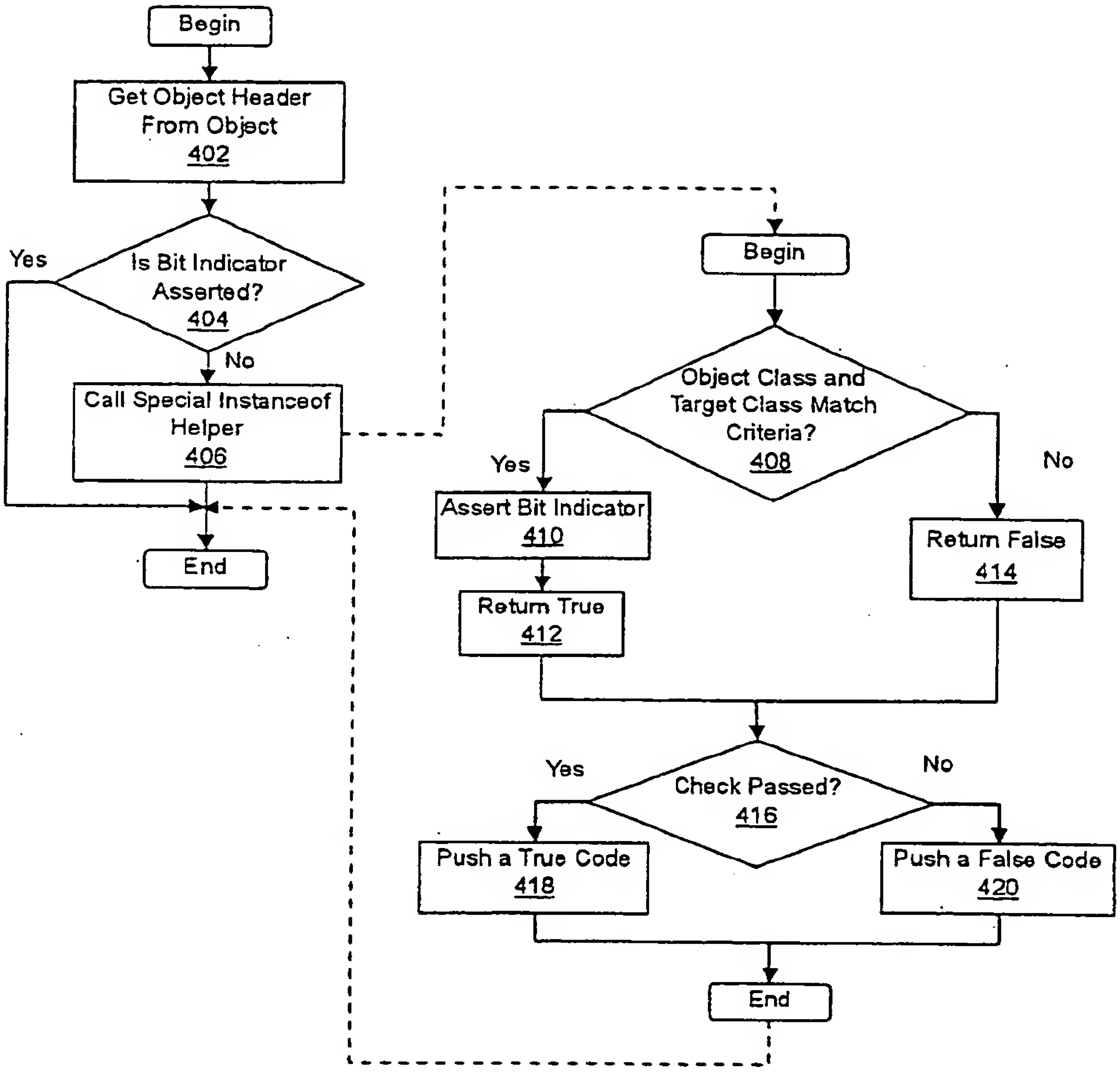
Begin
Get Object Header From Object 402
Is Bit Indicator Asserted? 404
Yes
No
Call Special Instanceof Helper 406
End
Begin
Object Class and Target Class Match Criteria? 408
Yes
No
Assert Bit Indicator 410
Return True 412
Return False 414
Check Passed? 416
Yes
No
Push a True Code 418
Push a False Code 420
End

FIG. 4

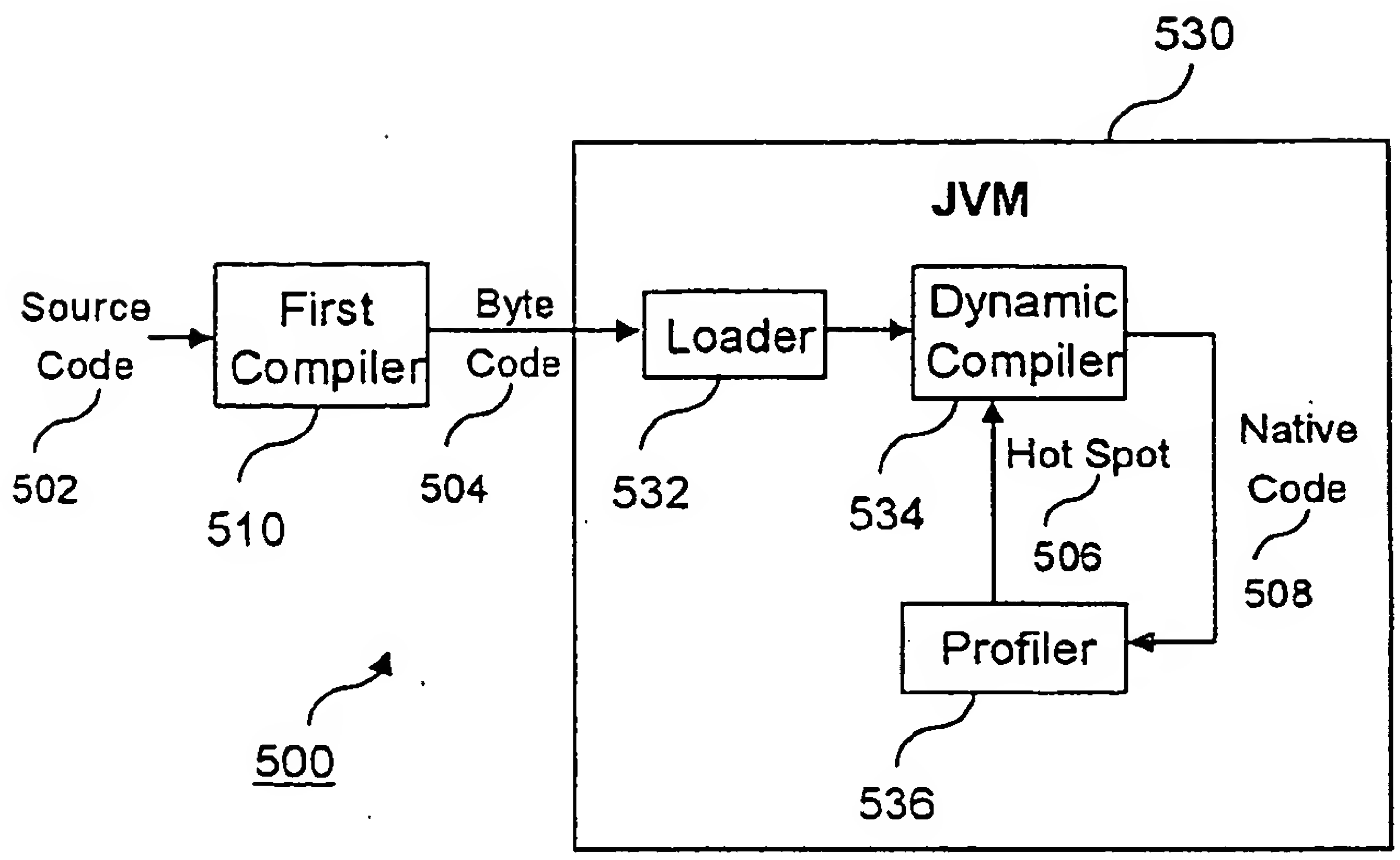
530
JVM
Source Code
502
First Compiler
510
Byte Code
504
Loader
532
Dynamic Compiler
534
Hot Spot
506
Native Code
508
Profiler
536
500

FIG. 5

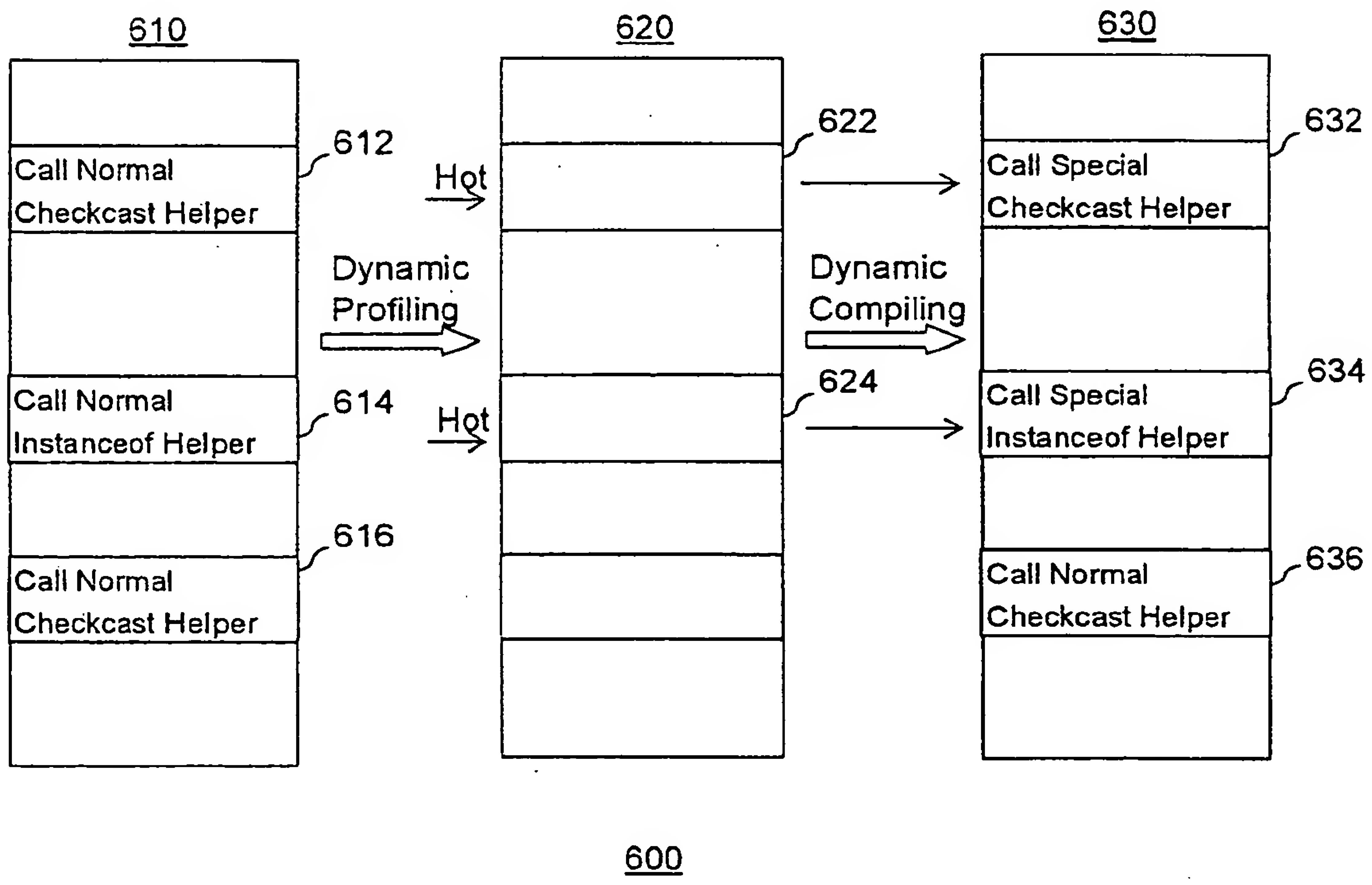
610
620
630
Call Normal Checkcast Helper
612
Hot
622
Call Special Checkcast Helper
632
Dynamic Profiling
Dynamic Compiling
Call Normal Instanceof Helper
614
Hot
624
Call Special Instanceof Helper
634
Call Normal Checkcast Helper
616
Call Normal Checkcast Helper
636
600

FIG. 6